1997

Evaluating the Performance of Distributed Architectures for Information Retrieval using a Variety of Workloads

Brendon Cahoon
University of Massachusetts - Amherst

Kathryn S. McKinley
University of Massachusetts - Amherst

Evaluating the Performance of Distributed Architectures for Information Retrieval using a Variety of Workloads *

Brendon Cahoon Kathryn S. McKinley
University of Massachusetts at Amherst

Abstract

Information explosion across the Internet and elsewhere offers access to an increasing number of document collections. In order for users to effectively access these collections, information retrieval (IR) systems must provide coordinated, concurrent, and distributed access. In this paper, we describe a fully functional distributed IR system based on the Inquery unified IR system. To refine this prototype, we implement a flexible simulation model which we use to present a series of experiments using a variety of workloads that measure system performance. We vary numerous system parameters such as the number of users, document collections, terms per query, query term frequency, think time, answers returned, and workload. Based on our initial results, we recommend simple changes to the prototype and evaluate the changes using the simulator. Because of the significant resource demands of information retrieval, it is not difficult to generate workloads that overwhelm system resources regardless of the architecture. However under some realistic workloads, we demonstrate system organizations for which response time gracefully degrades as the workload increases and performance scales with the number of processors. This scalable architecture includes a surprisingly small number of brokers through which a large number of clients and servers communicate.

Categories and Subject Descriptors: C.2.4 [**Computer-Communication Networks**]: Distributed Systems–distributed applications; C.4 [**Performance of Systems**]: Performance Attributes; H.3.4 [**Information Storage and Retrieval**]: Systems and Software;

General Terms: Experimentation, Performance

Additional Key Words and Phrases: Distributed information retrieval architectures

*This material is based on work supported by the National Science Foundation, Library of Congress, and Department of Commerce under cooperative agreement number EEC-9209623. This work is supported in part by NSF Multimedia award CDA-9502639 . Kathryn S. McKinley is supported by an NSF CAREER Award CCR-9624209. Any opinions, findings, and conclusions or recommendations expressed in this material are the authors and do not necessarily reflect those of the sponsors. Authors' addresses: B. Cahoon, Department of Computer Science, University of Massachusetts, Amherst, MA, USA; email: cahoon@cs.umass.edu; K.S. McKinley, Department of Computer Science, University of Massachusetts, Amherst, MA, USA; email: mckinley@cs.umass.edu.

1 Introduction

The increasing number of large, unstructured text collections require full-text information retrieval (IR) systems in order for users to access them effectively. Current systems typically only allow users to connect to a single database either locally or perhaps on another machine. A distributed IR system should be able to provide multiple users with concurrent, efficient access to multiple text collections located on remote sites. Since the documents in unstructured text collections are independent, IR systems are ideal applications to distribute across a network of workstations. However, the high resource demands of IR systems limit their performance, especially as the number of users, as well as the size and number of text collections, increases. Distributed computing offers a solution to these problems. Systems based on distributed architectures use resources more efficiently and in parallel by spreading work across a network of workstations.

Only recently have people started to build distributed architectures for information retrieval. Several researchers have created distributed IR systems and they have demonstrated the feasibility of distributed architectures for information retrieval [Harman et al., 1991, Macleod et al., 1987]. However, it is not clear from these initial implementations how the systems will perform in practice. The focus of this paper is to design distributed information retrieval architectures by analyzing the performance of potential systems under a variety of workloads. We begin with a prototype implementation of a distributed information retrieval system using Inquery; an inference network, full-text information retrieval model [Callan et al., 1992]. Our system adopts a variation of the client-server paradigm that consists of clients connected to information retrieval engines through a central administration broker, as we illustrate in Figure 1.

In our prototype, we use the Inquery system as the retrieval engine. In the original Inquery system (not distributed), clients specify an Inquery database, connect to it, interact with it, and finally disconnect. In the prototype distributed system, clients search multiple databases simultaneously. To build our prototype, we made the fewest possible changes to the underlying software. We created a single central broker, which we call the *connection server*, to administrate connections between clients and the IR engines. We refer to the IR engines as *Inquery servers*. The connection server maintains a list of available collections and their locations and brokers all of the clients' retrieval requests and Inquery server responses. We describe this distributed system in detail in Section 2. We measure the system and use it to drive a simulator in which we can easily move and replicate functionality to investigate alternative architectures for our distributed system. Section 3 presents this simulation model. We validate our model by comparing the performance of the simulator to the actual system for query operations.

We parameterize our simulation model by system features such as CPU, disk, and network demands.

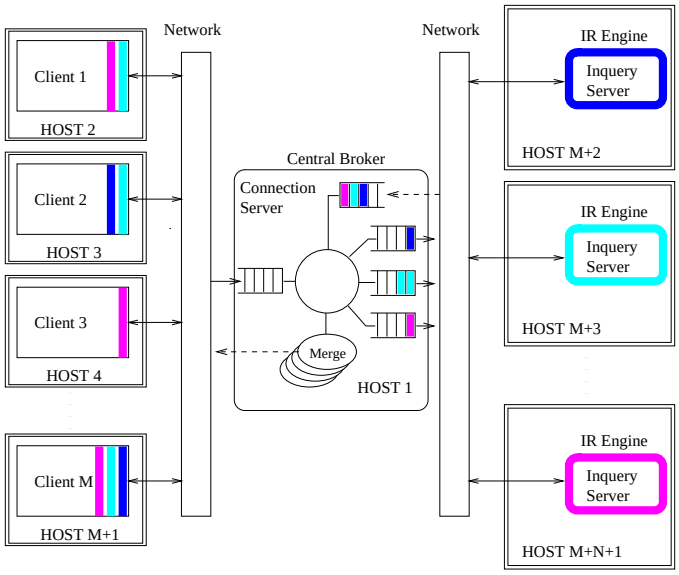

Figure 1: Our Distributed Information Retrieval System

This model allows us to investigate systems that vary from our implementation. In Section 4, we measure system performance including response time and resource utilization for a variety of configurations. During our investigation we identify bottlenecks and study the effects of various architectures and parameters. Our goal is to use resources efficiently by maximizing parallelism and ensuring scalability. We also maintain the effectiveness, in terms of recall and precision [Callan et al., 1995a], of a stand-alone IR system.

Clients model the activities of realistic users during our experiments. Each client issues query evaluation commands, obtains summary information for query results, and retrieves documents. Clients also simulate think time after each command. The different IR commands have distinct effects on performance. We show the architecture is able exploit parallelism in the summary information commands which improves response times as the size of the architecture increases, up to 8 or 32 Inquery servers.

We first investigate an architecture that distributes a single text collection among the available machines. We fix the size of the single text collection regardless of the number of machines. Thus, clients always search a fixed amount of text. The single text collection architecture is scalable for small configurations when there is a balance between the number of clients and Inquery servers. However, the architecture does not scale well and performance degrades for large configurations. The system achieves the best performance with 8 Inquery servers.

We also explore an architecture that distributes multiple, independent text collections. As the number of machines increases, the amount of text to search also increases. We believe the multiple text collection architecture is more applicable to actual systems, especially for large architectures, and we present more

detailed experiments. We vary the number of clients and text collections, the number of terms per query, the query term frequencies, the number of answers returned, the amount of think time, and the number of summary/document retrieval operations. In order to generalize our results, we experiment with a range of values for these parameters and discuss their effect on system performance. When clients issues short queries, response times are reasonable for many different numbers of clients and Inquery servers. However, the connection server is a bottleneck for a large number of clients and Inquery servers. The architecture does not perform well when users enter long queries due to bottlenecks in the Inquery servers. Interestingly, our results show that system performance improves as we increase the number of text collections to search, up to 8 or 32 collections, due to parallelism from the summary information operations.

Our results show that the connection server is often a bottleneck, especially when clients enter short queries. We concentrate on alleviating this bottleneck since users typically enter short queries [Croft et al., 1995]. We show that adding a small number of brokers to manage the clients and Inquery servers is an effective strategy for improving performance and ensuring scalability for short queries. We also experiment with moving the merging functionality from the connection server to the clients, but this architecture change does not improve performance. Section 5 compares our work to previous work and Section 6 summarizes our results.

2 A Distributed Information Retrieval System

This section describes the implementation of our distributed IR system. As shown in Figure 1, the distributed system consists of a set of clients, a connection server, and a set of Inquery servers. The distributed system enables multiple, simultaneous connections between clients and Inquery servers. The different components of the architecture communicate using a local area network. Each component may reside on a different host and operates independently of the others. In this section, we describe the functionality and interaction between the clients, the connection server, and the Inquery servers.

2.1 Clients

The clients are lightweight processes that provide a user interface to the retrieval system. Clients interact with the distributed IR system by connecting to the connection server, as illustrated in Figure 1. The clients initiate all work in the system, but the perform very little computation. The clients can issue the entire range of IR commands but, in this paper, we focus on *query*, *summary information*, and *document retrieval* commands.

A client sends query commands to the connection server. A **query command** consists of a set of words or phrases (terms). The command either specifies the list of Inquery servers to search or the client allows the connection server to determine the appropriate collections to search. In our experiments, we assume the clients choose the Inquery servers. Query responses consist of a list of *n* document identifiers ranked by belief values which estimate the probability that the document satisfies the information need. The user specifies an appropriate value for *n* or the system returns a default number of answers. We experiment using different values for *n*.

Summary information consists of the title and the first few sentences of a document. The **summary information command** consists of a set of document identifiers and their collection identifiers. The user may specify the number of summary entries to obtain but, in our experiments, each client obtains summaries for 15 documents at a time.

Clients may also retrieve complete documents by sending a **document retrieval command** to the connection server. The command consists of a document identifier and collection identifier. In response, the connection server returns the complete text of the document from the appropriate Inquery server.

A client issues a command and waits for the connection server to return the results before it issues another command. Users issue queries and document commands. A client automatically issues the first summary information command when it receives a query response. A client issues additional summary information commands at the user's request.

2.2 Connection Server

The clients and Inquery servers communicate via the connection server. The connection server is a lightweight process that keeps track of all the Inquery servers, outstanding client requests, and organizes responses from Inquery servers. The connection server continuously polls for incoming messages from clients and Inquery servers. The connection server handles outstanding requests from multiple clients.

A client sends a command to the connection server which forwards it to the appropriate Inquery servers. In our experiments, the command indicates the Inquery servers. Since each Inquery server only processes one request at a time, the connection server maintains a queue of outstanding requests for each Inquery server as illustrated in Figure 1. The connection server inserts commands from clients onto a queue if an Inquery server is currently processing a command from another client. When the connection server receives an answer from an Inquery server, it forwards the next command on the corresponding queue to the Inquery server.

The connection server maintains intermediate results for commands specifying multiple Inquery servers.

When Inquery servers return results, the connection servers merges them with other results. After all the Inquery servers involved in a command return results, the connection server sends a final result to the client. Only query and summary commands may specify multiple Inquery servers. For a query command, each Inquery server sends its top *n* responses back to the connection server. The connection server maintains a sorted list of the overall top *n* entries until all the Inquery servers respond. The connection server merges new results with the existing sorted list. We assume the relative rankings between documents from independent collections are comparable, but this assumption is clearly tenuous. For example, one collection may be irrelevant to a particular query, but if the user includes it, the overall response may still include its top ranked responses. Other research is investigating techniques to automatically select appropriate collections with respect to specific queries and merge results [Callan et al., 1995b, Voorhees et al., 1995, Viles and French, 1995, Moffat and Zobel, 1995]. For a summary command, the connection server must read the message to determine which Inquery servers contain the appropriate documents and then send messages to each Inquery server to acquire the summary information. The connection server merges the summary information responses and sends a single message back to the client with the summary information for all the documents. The connection server does not maintain intermediate results for document retrieval commands; it simply forwards a document as soon as the Inquery server sends it.

2.3 Inquery Servers

The Inquery server uses the Inquery retrieval engine to provide IR services such as query evaluation and document retrieval. Inquery is a probabilistic retrieval model that is based upon a Bayesian inference network [Callan et al., 1992]. Inquery accepts natural language or structured queries. Internally, the system stores text collections using an inverted file. Previous work demonstrates that Inquery is an effective retrieval system for large, full-text databases [Callan et al., 1995a].

The Inquery server accepts a command from the connection server, processes the request, and returns the result back the connection server. The Inquery server only processes one command at a time.

3 Simulation Model

In this section, we present a simulation model for exploring distributed IR system architectures. Simulation techniques provide an effective and flexible platform for analyzing large and complex distributed systems. We can quickly change the system configuration, run experiments, and analyze results without making numerous changes to large amounts of code. Furthermore, simulation models allow us to easily create very

large systems and examine their performance in a controlled environment.

To implement the simulator, we use YACSIM, a process oriented discrete event simulation language [Jump, 1993]. YACSIM contains a set of data structures and library routines to manage user created processes and resources. Its process oriented nature enables the structure of the simulator to closely reflect the actual system.

Our simulation model is simple, yet contains enough details to accurately represent the important features of the system. Section 2 describes our model's basic architecture and functionality. The hardware resources we model include CPUs, disks, and the LAN. We model the clients, Inquery servers, and connection servers as different processes. Processes simulate the activities of the real system by requesting services from resources. The simulator is driven by empirical timing measurements obtained from our prototype.

Our technique for designing an environment for studying distributed information retrieval architectures is similar to the model that Brumfield *et al.* present [Brumfield et al., 1988]. However, they simulate a distributed object-oriented database system while our work focuses on IR systems. We configure a simulation by defining the architecture of the distributed IR system using a simple command language. A configuration file contains the commands which the simulator reads at start-up time.

3.1 System Measurements

To accurately model an IR system, we analyze the prototype distributed Inquery system and measure the resources used for each operation. We focus on CPU, disk, and network resources and we do not measure memory and cache effects. Empirical measurements rather than an analytical model drive the activities performed in the simulator. The simulator uses the following parameters: query evaluation time, document/summary retrieval time, connection server time, network latency, and time to merge results. We obtained measurements of the parameters from the prototype system using Inquery version 2.1 running on a DECsystem-5000/240 (MIPS R3000 clocked at 40 MHz) workstation running Ultrix V4.2A (Rev. 47) with 64 MB of memory and 300 MB of swap space. We instrument the Inquery system to report time measurements during run time using the `ftime()` and `getrusage()` system calls. To obtain TCP network performance, we use TTCP from the US Army Research Laboratory.

We examine several different text collections and query sets to obtain system measurements. We examine TIPSTER 1, a large heterogeneous collection of full-text articles and abstracts [Harman, 1992], a database containing the Congressional Record for the 103rd Congress [Croft et al., 1995], and a small collection of abstracts from the *Communications of the ACM* [Fox, 1983].

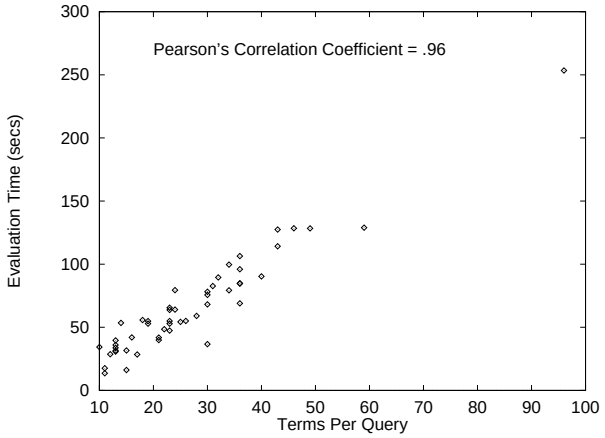

Figure 2: Query Length vs. Evaluation Time

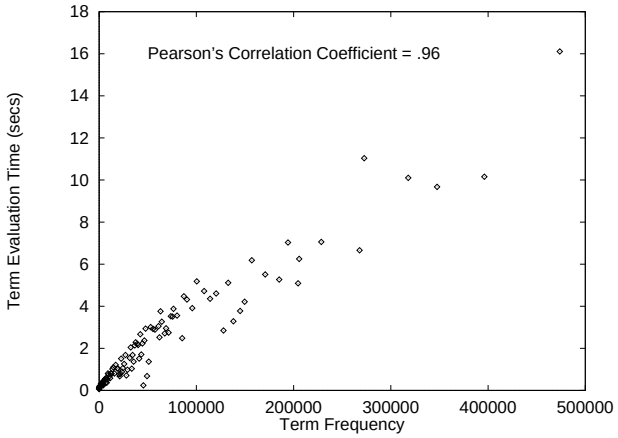

Figure 3: Term Frequency vs. Evaluation Time

Query Evaluation Measurements

The simulator uses a simple, yet accurate model to represent query evaluation time. Based upon our measurements on Inquery using our query sets, evaluation time is very strongly related to the number of terms per query and the frequency of each of the terms. Using the TIPSTER 1 query set, Figures 2 and 3 illustrate the correlation of terms per query and query term frequency to evaluation time, respectively. Pearson's correlation coefficient for both sets of data is 0.96 which indicates a strong positive relationship.[1]

Our query evaluation model is a function of the number of terms per query and the frequency of the individual query terms plus a small overhead

$$\begin{aligned} time &= \sum_{i=1}^{n} eval\_term\_time(term_i) \\ overhead &= n \times time \times 0.015 \\ eval\_query\_time &= time + overhead \end{aligned}$$

where n is the number of terms in the query and $term_i$ is the i^{th} term.

We model *eval_term_time* as an increasing linear function of the term frequency. To obtain this function we use the TIPSTER 1 query set and text collection to measure the evaluation time for terms of different frequencies. Figure 3 shows this data. We create *eval_term_time* by fitting the data in Figure 3 with three straight lines using a least square method. The first line covers term frequencies up to 10000, the second lines covers term frequencies 10000-100000, and the third lines covers term frequencies after 100000. The

[1]Pearson's correlation coefficient ranges between -1 and +1. Values -1 and +1 indicate a perfect linear relationship and occur when all points lie on a downward or upward sloping line, respectively. A value of 0 indicates there is no linear relationship.

model is more accurate using three lines rather than a single straight line for all the data.

Using this function, the time to evaluate a single term ranges from 0.12 seconds for a term appearing once to 16.5 seconds for a term appearing 554,568 times (the maximum term frequency in TIPSTER 1). We divide the evaluation time into CPU and disk access time. Disk access time account for 11% to 33% of total evaluation time. The percentage of disk access time rapidly decreases as the term frequency increases (disk access time is 14% for a term occurring 100000 times). The overhead figure accounts for the time Inquery spends combining the results of each of the terms. We empirically determine the overhead amount by measuring the evaluation time for queries of different lengths. Without the overhead amount, we underestimate the evaluation time of long queries.

The query evaluation model is slightly different than the model we used in previous work [Cahoon and McKinley, 1996]. The new *eval_term_time* function is more accurate and we did not include any overhead in the old model. In practice our simulator overestimates query evaluation times since our model assumes a query command returns the entire list of relevant answers. In our experiments, the Inquery servers only return the top *n* answers. For small values of *n* the actual system performs slightly less processing. We do not include this effect in our model since it is difficult to characterize.

Document Retrieval Measurements

We measure Inquery to determine the amount of time it takes to retrieve a document. For our text collections, the retrieval time is variable and there is not a strong correlation between document size and retrieval time. The low correlation is due to the size of the documents in our text collections which are not very large so retrieval occurs very quickly. In our collections, the average size of a document in the TIPSTER 1, Congressional Record, and the CACM is 2.3 KB, 11.7 KB, and 0.5 KB, respectively.

The simulator represents the document retrieval time for an Inquery server as a constant value, 0.31 seconds, which is the average document retrieval time for 2000 randomly selected documents from the TIPSTER 1 collection. The simulator represents retrieval time as CPU processing plus disk access time. Our measurements show that disk access time accounts for 84% of the total retrieval time on average. Thus, the simulator represents CPU processing as 0.05 seconds and disk access time as 0.26 seconds. To ensure our simulator is accurate for our experiments, the average document size returned by an Inquery server is the same as the average size in the TIPSTER 1 collection, 2.3 KB. The simulator also uses the document retrieval time to compute the summary information retrieval time. We implement the summary information operation as a series of document retrieval operations. For each summary entry, an Inquery server reads a complete document. However, the Inquery servers only return the summary portion of the document. In our

experiments, the average size of a document summary is 120 bytes.

Connection Server Time

The connection server time consists of two values; the processing time for handling a message and the time to merge results. We obtain the message handling time by measuring the prototype connection server. When the connection server receives a message from either a client or Inquery server, the simulator uses a constant value, 0.1 CPU seconds, to represent the message processing time. The time to merge query results depends upon the number of answers an Inquery server returns. For example, our measurements show that merging a list with 100, 1000, and 2000 answers takes 1.9, 17.9, and 35.7 milliseconds on average, respectively.

Network Time

We represent network time as sender overhead, receiver overhead, and network latency. The sender and receiver overhead is the CPU processing time for adding and removing a message from the network. The network latency is the amount of time the message spends on the network itself. These times depend upon the size of the message and the bandwidth of the network. We obtain the sender and receiver overhead times using TTCP by measuring the time to send messages between two DECsystem-5000 workstations connected by a lightly loaded 10Mbps Ethernet. The following table shows different network times for 32 B, 1 KB, and 10 KB messages.

Category	Message Size		
	32 B	1 KB	10 KB
Sender Overhead	0.07 ms	0.14 ms	2.9 ms
Receiver Overhead	0.1 ms	0.25 ms	1.8 ms
Network Latency	0.0256 ms	0.82 ms	8.2 ms

3.2 Validation

We validate the simulator's query evaluation times against the actual implementation using a configuration consisting of single client, Inquery server, and connection server. Each of the components runs on a separate host. We do not validate the document retrieval times since our measurements show the actual document retrieval time is variable due to the quick access times. As we mention in Section 3.1, the time to retrieve documents in the simulator is a constant value.

We validate our simulator by creating artificial queries and comparing the results from running the queries on the actual system to the results from our simulator. We randomly generate queries which cor-

respond to the types of queries our simulator generates during our experiments (see Section 3.3). The two parameters we use to generate queries are the number of terms per query and the query term frequencies. We generate 100 short queries, 100 medium length queries, and 100 long queries. We do not generate queries with multiple occurrences of the same term since our model does not account for these types of queries. In previous work, we used a different validation method which is less thorough and realistic [Cahoon and McKinley, 1996]. Although our old model is valid, we have slightly improved our query evaluation model to be more accurate.

We show our validation results using the short, medium, and long generated queries in the table below. The second column shows the average percentage difference between the simulator and the actual system. On average, the simulator is 0.7% slower than the actual system and the the standard deviation is 9.9%.

Difference Between Simulated and Actual Times						
Query Type	% Difference Avg.	Std	±10%	±15%	±20%	±25%
Short	-3.3%	11.1%	72%	88%	93%	100%
Medium	3%	7.8%	78%	91%	97%	100%
Long	-1.9%	9.3%	74%	88%	96%	100%
All	-0.7%	9.9%	74.7%	89%	95.3%	100%

Columns 4–7 breakdown the amount of variation in more detail. We show the percentage of queries running on the simulator that fall within ±10%, ±15%,±20%, and ±25% of the actual system. For example, column 4 shows the simulator query evaluation time for 74.7% queries fall within 10% of the actual system. The last column shows the simulator evaluates all the queries within 25% of the actual system time.

The validation results show that our simple query evaluation model accurately reflects the actual system. In general, the simulator matches the actual system very closely, but our simulator is not able to accurately model every query. Due to our simple model it is difficult to reduce the amount of variation in the difference between our simulator and the actual system. Given that our simulator evaluates most queries very accurately, it is not worth the added complexity to change the model.

3.3 Experimental Parameters

In this section, we describe the parameters we use in our simulation experiments. Table 1 presents the parameters, their values, and abbreviations. We describe each parameter in more detail below.

Number of Clients/Inquery Servers (C/IS) We experiment with both small and large system configurations. Measuring the effect of increasing the number of clients and Inquery servers provides insight into

Parameters	Abbreviation	Values
Clients	C	1 4 8 32 64 128 256
Inquery Servers	IS	1 4 8 32 64 128
Terms per Query (average) shifted neg. binomial dist.	TPQ	2 12 27
Query Term Frequency dist. from queries	QTF	Obs. Dist. Low Skew High Skew
Answers Returned constant values	AR	100 1000 2000
Think Time/Summary normal dist. (μ,σ)	TTS	7,1.5 15,3 30,6 60,12
Think Time/Document normal dist. (μ,σ)	TTD	15,3 30,6 60,12 180,36
Documents Retrieved uniform dist.	DR	1–5 8–12 15–20
Summary Operations uniform dist.	SO	1–5 8–12 15–20

Table 1: Experimental Parameters

distributed IR architectures by identifying bottlenecks and helping to understand system utilization and scalability. We experiment with each combination of clients and Inquery servers listed in Table 1. The largest configuration consists of 256 clients and 128 Inquery servers.

Terms Per Query (TPQ) Table 1 shows the three different average query lengths we use in our experiments. We obtain two of the values from the 103rd Congressional Record (2 terms) and TIPSTER 1 query sets (27 terms). We also use an intermediate value (12 terms) since the two query sets have very different characteristics. Figures 4 and 5 show the query length distributions for the 103rd Congressional Record and TIPSTER 1 query sets. A shifted negative binomial distribution closely matches the distributions in our query sets. The characteristics of a shifted negative binomial distribution are the number of trials, *n*, the probability of success, *p*, and the amount of shift, *s*. The following table shows the values we use in our experiments. Wolfram also uses a shifted negative binomial distribution to model terms per

Description	Query Set	Avg Length	*n*	*p*	*s*
Short Queries	103rd CR	2	4	0.8	1
Medium Queries	N/A	12	2	0.77	5
Long Queries	TIPSTER 1	27	2	0.1	10

query [Wolfram, 1992]. However, we use slightly different values for *n*, *p*, and *s* in order to obtain better estimates of our data sets.

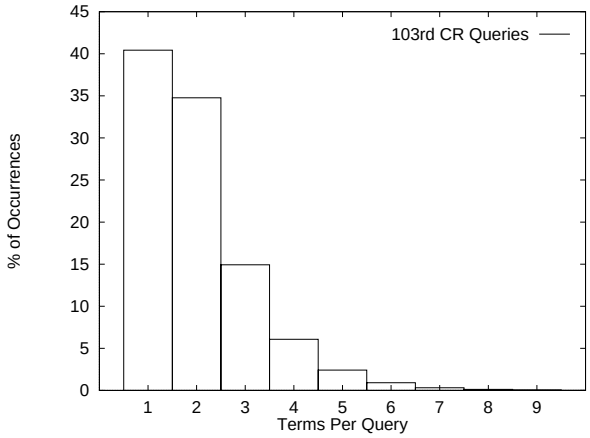

Figure 4: 103rd CR Query Lengths

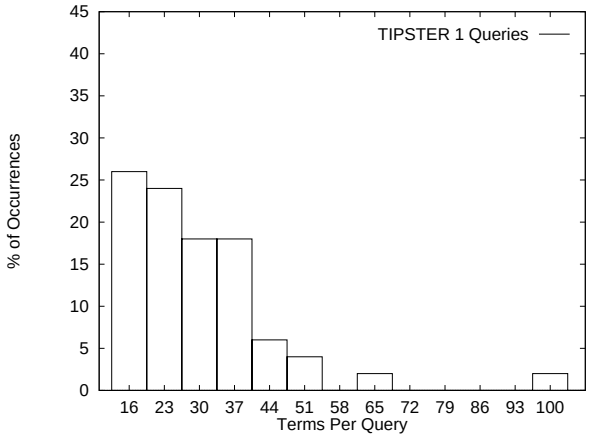

Figure 5: TIPSTER 1 Query Lengths

Distribution of Terms in Queries (QTF) Zipf documented the widely accepted distribution of term frequencies in text collections [Zipf, 1949]. In contrast, researchers do not agree on a commonly accepted distribution for term frequencies in queries [Wolfram, 1992]. Figure 6 shows the distribution of our query sets. The query term frequency distributions for the query sets are similar but the distributions do not closely match a well known distribution. In our experiments, we use the distribution of query term frequencies from the TIPSTER 1 query set. We call this our *observed* query term frequency distribution. We also use a distribution that is skewed towards terms occurring less frequently and a distribution that is skewed towards terms occurring more frequently. We shift the observed frequency to the left by 85% to create the low frequency distribution. To create the high frequency distribution, we shift the observed frequency to the right by 100%

Number of Answers Returned (AR) For each query, the IR system returns a sorted list of matching documents to the clients. The list contains document identifier numbers instead of the complete text. The number of answers returned affects network traffic and processing by the connection servers. The three different constant values that we use are 100, 1000, and 2000 answers.

Think Time (TT) In the simulated workload, clients "think" after receiving summary information and documents. This value accounts for the time, in seconds, that users look at the results of their requests. We model think time as a normal distribution. When examining our results, it is important to note that think time can be large in comparison to the time the system takes to perform requests. Since we do not have statistics representing actual user think times, we use four different distributions ranging from a small amount of think time to a large amount. Table 1 lists the mean and standard deviation for the think time distributions. Further reducing think time or adding clients have similar effects on performance in this system.

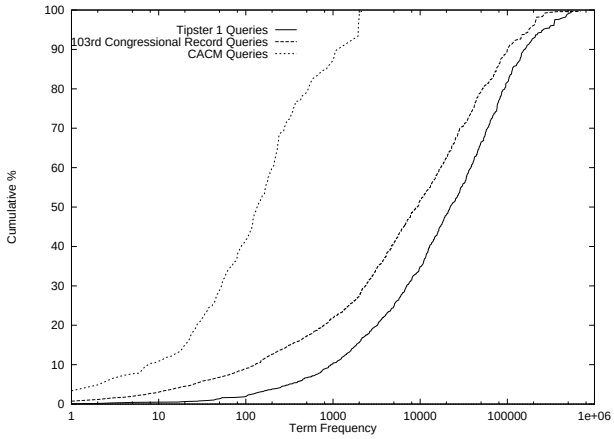

Figure 6: Query Term Frequency Distributions

Document Retrieval/Summary Information (DR/SO) We vary the number of summary and document retrieval operations that each client performs. The entries in Table 1 represent a range of values from which the simulator randomly chooses values. A single summary information operation retrieves entries for 15 documents. The simulator generates different document lengths from the distribution of document lengths in the TIPSTER 1 collection. The document length ranges from 0.24KB to 12KB with an average of 2.3KB. The number of summaries and the document size determine the time to send results across the network.

3.4 Workload

The workload consists of the basic retrieval operations described in Section 2: query evaluation, obtaining summary information, and document retrieval. The simulator does not model more complicated functions such as relevance feedback. The simulator only models natural language queries and does not evaluate structured query operations such as phrase and proximity operators. In the simulator, clients repeatedly perform the following **transaction sequence**: *evaluate a query, obtain summary information of top ranking documents, think, retrieve documents, think.* During each transaction sequence the clients may issue multiple summary information and document retrieval commands. We simulate thinking time after each command. The simulator varies the specific operations for each client and during each sequence. For example, the model generates new queries and retrieves a different number of documents for each iteration.

3.4.1 Simulation Output

During each simulation execution, we measure different performance statistics such as the average transaction sequence time, connection server utilization, queue lengths, network utilization, Inquery server uti-

lization, and response time. For each series of experiments, we display the corresponding values of the parameters and the abbreviations listed in Table 1.

4 Experiments and Results

In this section, we evaluate the performance of our distributed information retrieval architecture and present the results.

In Section 4.1, we show the effect of equally distributing a single text collection among each of the Inquery servers. This architecture models a distributed system that maintains a single large database but exploits parallelism by operating independently on each Inquery server. For this experiment, we present results showing the effect of terms per query on performance. These results show that this architecture performs well for small configurations when there is a balance between the number of clients and Inquery servers. The best transaction sequence time occurs for a configuration with 8 Inquery servers. Performance degrades, however, as the size of the architecture grows.

In Section 4.2, we show an architecture that manages multiple, distinct text collections. In this case, each Inquery server maintains a different database and the clients evaluate queries on a subset of the available databases. We vary parameters such as the number terms per query, query term frequency, answers returned, think time, and the number of summary and document operations per transaction sequence to provide us with a deeper understanding of the system performance under different workloads. We present more detailed results using the multiple text collection architecture since we believe this architecture more closely resembles actual implementations of large distributed architectures. The performance trends for the multiple text collection and single text collection architectures are very similar. We show that performance improves as we increase the number of Inquery servers, up to 8 or 32 servers, even though clients search more information. The improvement is due to parallelism in the summary information commands.

Running experiments with a variety of parameters allows us to gain a better understanding of system performance. For example, we demonstrate that for short, realistic queries, several architecture configurations scale with the number of processors and degrade gracefully as the number of clients (work) increases. Our results illustrate that the system achieves good performance under varying conditions if we can maintain a balance between connection server and Inquery server utilization. Some conditions do cause the system performance to rapidly deteriorate. For example, the architecture does not perform well for very large configurations or when either the utilization of the connection server or Inquery servers is high. We often see poor utilization for long user queries.

Varying query term frequency changes the Inquery server utilization when the Inquery servers are the bottleneck. Unfortunately, the overall system performance is still poor for large configurations. We show that varying the number of answers returned affects network utilization, but since network utilization is so low, we see little change in performance. The amount of think time affects system performance for a small number of clients, but the effect decreases as the number of clients increases. Our results show that increasing the number of summary and document operations per transaction sequence improves response time since summary and document operations are less computationally intensive than query operations.

In Sections 4.3 and 4.4, we change the architecture in order to improve the performance of our distributed system. Since users tend to issue short queries in practice, we concentrate on eliminating the bottleneck in the connection server. We experiment with adding connection servers to introduce more parallelism. We test configurations using two and four connection servers and find this is sufficient to relieve the connection server bottleneck for up to 256 clients and 128 Inquery servers. We also experiment with moving the response merging from the connection server to the clients. Unfortunately, this change does not improve performance because the the amount of time spent merging is small and the cost to send more messages decreases the benefit of moving the merging functionality.

Unless otherwise stated, in each of our experiments the clients, connection server, and Inquery servers operate as described in Section 2. We allocate each of the basic components in the distributed system to its own host. Each host contains its own processor, memory, and secondary storage. A local area network with a bandwidth of 10Mbps connects the machines. Each of the Inquery servers maintains a 1 Gigabyte database (except in the first experiment in Section 4.1 where we distribute a single 1 GB database). The clients start at the same time and issue 20 transactions during a simulation run. We also ran experiments using a staggered start but did not see any noticeable change in the results.

4.1 Distributing a Single Text Collection - Fixed Workload

In this section, we examine the performance of the system when we divide a single 1 GB text collection among all the Inquery Servers. The size of the text collection managed by each Inquery server depends upon the number of Inquery servers. For example, in a system with 64 Inquery servers, each collection is 16 MB. This architecture models a distributed system that maintains a single large database, but exploits parallelism by operating independently on each portion. In this configuration, each client performs a fixed amount of work (*i.e.*, a client always searches 1GB of information regardless of the number of Inquery servers). Each client evaluates a query on all the Inquery servers. For this experiment, we only present results showing the effect of the number of terms per query on performance.

Distributing a Single Text Collection

Short Queries

TPQ	QTF	AR	TTS	TTD	DR	SO
2	Obs.	1000	30	60	1–5	1–5

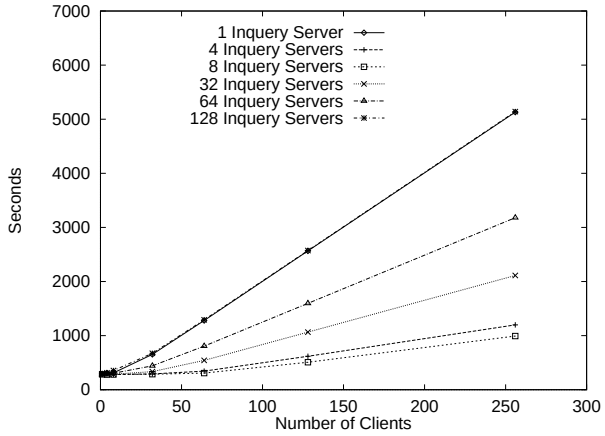

Figure 7: Average Transaction Sequence Time

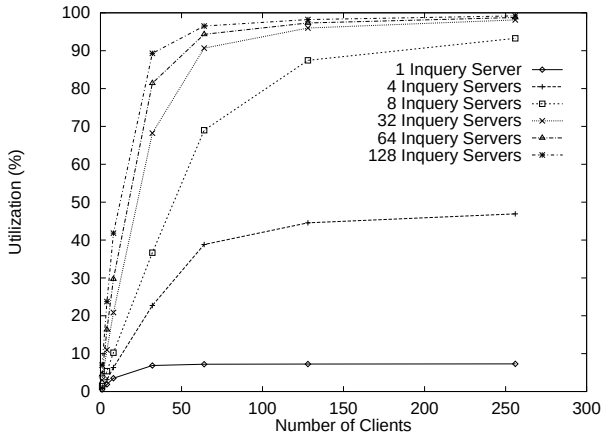

Figure 8: Connection Server Utilization

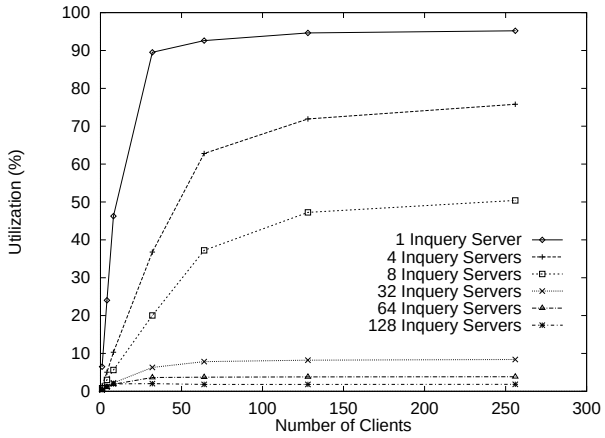

Figure 9: Average Inquery Server Utilization

Long Queries

TPQ	QTF	AR	TTS	TTD	DR	SO
27	Obs.	1000	30	60	1–5	1–5

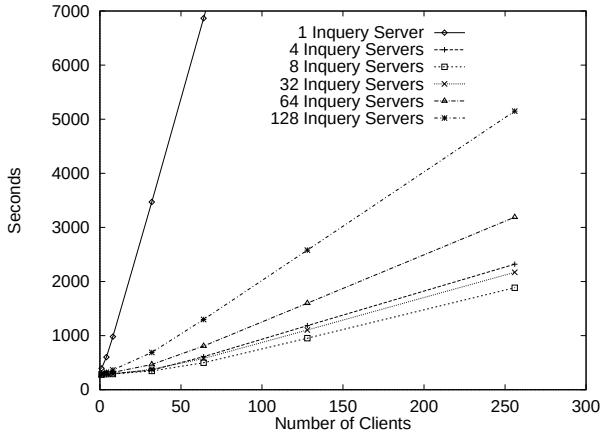

Figure 10: Average Transaction Sequence Time

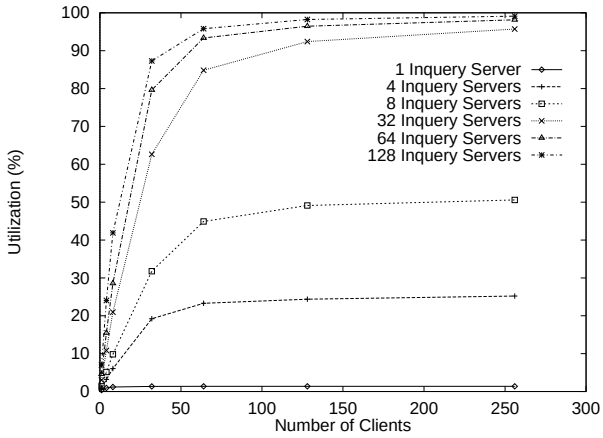

Figure 11: Connection Server Utilization

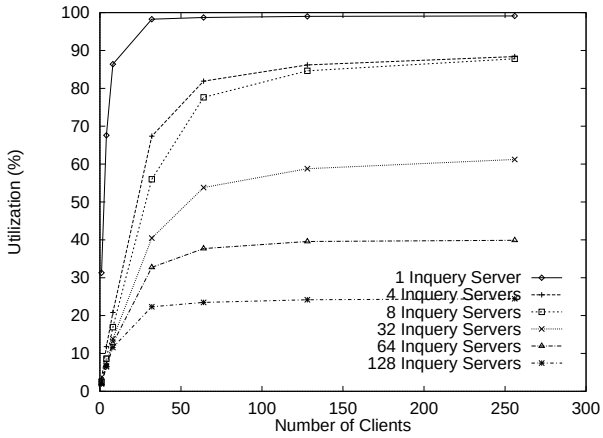

Figure 12: Average Inquery Server Utilization

4.1.1 Effect of Terms Per Query

In Figures 7–12, we present and compare the average transaction time, connection server utilization, and Inquery server utilization for short (Figures 7–9) and long (Figures 10–12) queries. In all figures, we display the number of clients, 1 to 256, on the x-axis. In Figures 7 and 10 we display the number of seconds on the y-axis. In Figures 8, 9, 11, and 12, we display process utilization on the y-axis.

Short Queries (TPQ=2) Figure 7 illustrates an improvement in the average transaction sequence time as we add Inquery servers and exploit parallelism, up to 8 Inquery servers.[2] Going from 1 to 8 Inquery servers improves performance for 256 clients by a factor of 5.17. However, when the system contains more than 8 Inquery servers, the performance degrades because the connection server becomes over utilized.

The performance improvement is due to a couple of factors. First, as we increase Inquery servers, the size of each database decreases which improves query evaluation time. For example, the system is able to search two 500 MB databases in parallel more quickly than searching a single 1 GB database. Second, more detailed measurements reveal some of the improvement stems from increased parallelism during summary retrieval. Recall that a single summary information operation retrieves 15 documents. A system with one Inquery server contains all the documents on the same machine. However, a system with multiple Inquery servers distributes the documents among the available Inquery servers. The 15 summary entries may reside on different Inquery servers resulting in parallel access of the summary information. In the best case, each of the 15 entries are located on different Inquery servers.

As Figure 7 shows, the average transaction sequence time degrades very slowly as we increase the number of clients for 4 or 8 Inquery servers. For example, for 8 Inquery servers as we increase the number of clients by a factor of 64 (from 1 to 64), system response time degrades by a factor of 1.08. However, the jump between 1 and 256 clients degrades performance by a less acceptable factor of 3.5. For 8 Inquery servers, the system achieves a good balance between connection server and Inquery server utilization.

The performance degradation for 32 or more Inquery servers occurs because the connection server becomes a bottleneck. Figure 8 shows the connection server utilization becomes very high for 32 to 128 Inquery servers. When the utilization exceeds 85%, the connection server does not process messages as quickly as the clients and Inquery servers send them. For example, the connection server's incoming queue length for utilization values greater than 85% exceeds 20 messages. Our results indicate the connection server effectively processes up to 8 requests per second. After this threshold, the connection server becomes over utilized. The bottleneck in the connection server explains the low utilization of the Inquery servers

[2] In this experiment, 1 and 128 Inquery servers have the same performance.

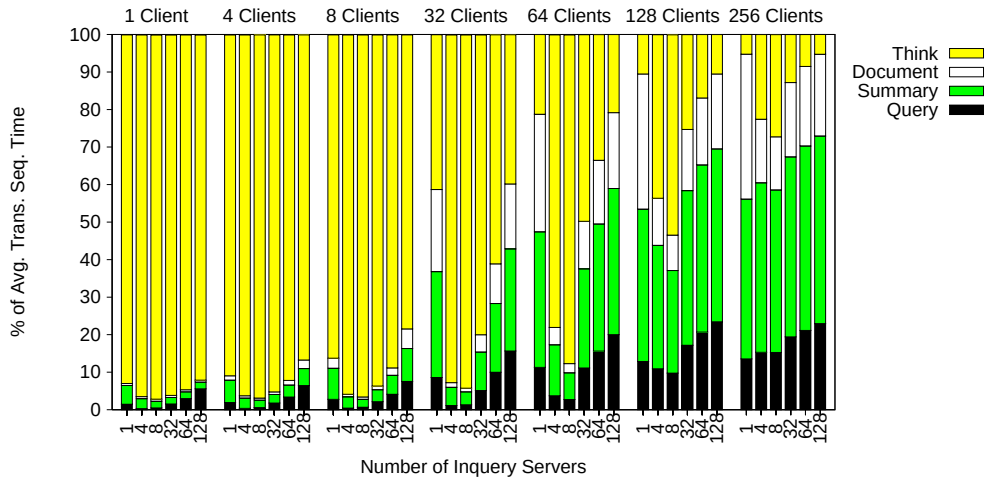

Figure 13: Time in Transaction Sequence

(Figure 9). The Inquery servers remain idle when the connection server is too busy to forward outstanding requests.

Figure 13 shows where time is spent during a transaction sequence as the number of clients and Inquery servers increase and clients issue short queries. Figure 13 divides the average transaction sequence time into time spent evaluating a query, obtaining summary results, obtaining documents, and time spent thinking. We normalize the values and display the times as a percentage of average transaction sequence time. The bar on the far left is for 1 client and 1 Inquery server and the bar on the far right is for 256 clients and 128 Inquery servers. For a small number of clients, most of the time in a transaction sequence is spent thinking. The average amount of think time per transaction sequence is 270 seconds for all client/Inquery server combinations. As the number of clients increases, the percentage of thinking time decreases significantly. It is important to note that the actual amount of time to evaluate a query by a single Inquery server and obtain summary information slightly decreases as the size of the architecture grows. Figure 13 shows that as the number of clients and Inquery servers increase, the system is unable to efficiently handle all the client's

No. of Clients	Command	Number of Inquery Servers					
		1	4	8	32	64	128
1	Query	4.23	0.86	1.38	4.39	8.39	16.37
	Summary	4.92	2.41	1.59	1.66	1.79	1.87
	Document	0.52	0.52	0.52	0.52	0.52	0.52
4	Query	5.9	1.04	1.58	5.11	10.08	20.0
	Summary	5.98	2.57	1.74	2.14	3.1	4.89
	Document	1.19	0.58	0.55	0.65	1.23	2.34
8	Query	8.55	1.23	1.83	6.15	12.49	6.75
	Summary	8.72	2.8	1.95	3.06	5.03	10.41
	Document	2.827	0.631	0.603	0.963	2.026	5.885

Table 2: Response Times for 1,4,8 Clients

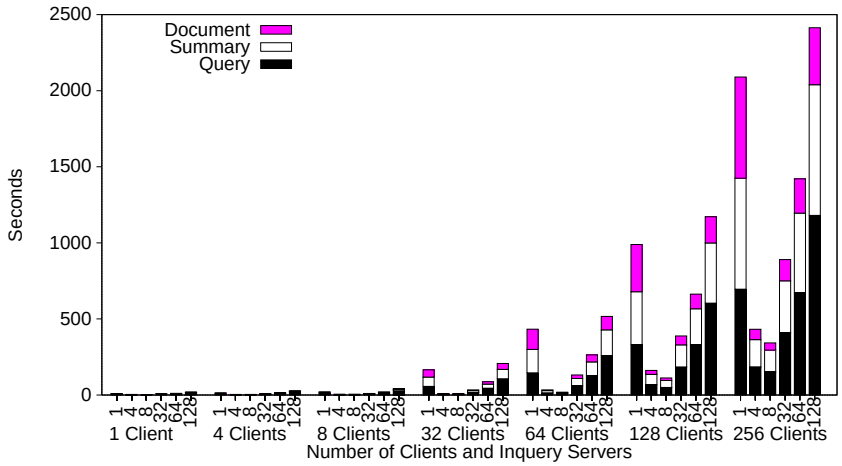

Figure 14: IR Command Avg. Response Time

requests because a request spends a large amount of time waiting to acquire resources.

Figure 14 shows average response times for query, summary, and document operations for all combinations of clients and Inquery servers when clients issue short queries. The difference between the values in Figures 14 and 7 is that the transaction sequence time in Figure 14 includes think time and multiple summary and document operations. We list response time values in Table 2 since it is difficult to see the actual values for 1, 4, and 8 clients.

Figure 14 and Table 2 illustrate that the system achieves good response times for a small number of clients and the system also performs well for several configurations involving 32, 64, and even 128 clients. Figure 7 reflects these encouraging response times in the average transaction sequence time. For example, the best response and transaction sequence times occur with 8 Inquery servers. For other configurations, the performance of our distributed architecture degrades due to high contention for resources. Table 2 also shows that obtaining summary information for 15 documents is a relatively expensive operation, especially with 1 Inquery server. The response time improves for 4 and 8 Inquery servers because the system is able to parallelize the summary command.

Long Queries (TPQ=27) Similar to the results using short queries, Figure 10 shows the average transaction sequence time improves as we add up to 8 Inquery servers and performance degrades for more than 8 Inquery servers. In comparison with the results for short queries (Figure 7), the system performance does not scale well as the number of clients increases. For long queries, as we increase the number of clients from 1 to 256 on a system with 8 Inquery servers the performance degrades by a factor of 6.8 (compare this with 3.5 for short queries).

The Inquery servers are initially the bottleneck when clients issue long queries. The bottleneck at the Inquery servers is responsible for the poor scalability as we add clients on a system with a small number of Inquery servers. The connection server quickly becomes the bottleneck as the system includes more Inquery servers. The bottleneck shifts quickly because each Inquery server is able to process queries more quickly as the size of each Inquery server's text collection decreases.

Medium Queries (TPQ=12) Similar to short and long queries, the system achieves the best performance using a configuration with 8 Inquery servers when clients issue medium length queries. Using medium queries, the architecture does not scale as well compared to short queries, but the performance is better than when clients issue large queries. For example, the average transaction sequence time for 256 clients and 8 Inquery servers is 4.4 times greater (1172 seconds) than 1 client and 8 Inquery servers (266 seconds). The corresponding value for short and long queries is 3.5 and 6.8, respectively. The system is unable to achieve a good balance of resources especially when the number of clients or Inquery servers is large. The connection server is the bottleneck for configurations consisting of more than 32 clients and Inquery servers. The Inquery server is the bottleneck for small configurations.

Summary Our results show it is difficult to achieve scalable performance with an architecture that distributes a single text collection. Regardless of query length, the best performance occurs on an system with 8 Inquery servers. The architecture achieves good performance for small configuration when there is a balance between the number of clients and Inquery servers. When an imbalance occurs, either the connection server or the Inquery servers are a bottleneck which results in poor performance. System utilization is the worst when clients issue long queries. In this case, the system rarely achieves a good balance of resources. We see that the architecture is able to effectively handle more users and Inquery servers as the query size decreases.

4.2 Multiple Text Collections - Scaled Workload

In this section, we measure the performance of a distributed IR system that maintains multiple text collections. In this configuration, each client selects a random subset of the available collections to search and searches half of the available collections on average. Each collection has an equal chance of being chosen. The workload increases both as a function of the number of Inquery servers and the number of clients (*i.e.*, the total amount of information to search increases with the number of Inquery servers). This workload mimics the scenario when the connection server automatically selects an appropriate subset of the available

collections to search. It also models the situation when the user manually chooses a subset of the collections to search.

In the following sections, we present results showing the effect of our experimental parameters on the performance of the architecture. These parameters include the number of users and text collections, terms per query, query term frequency, answers returned, think time, and number of summary and document operations. We list the values for each parameter in a table at the top of the figures which appear in this section.

4.2.1 Effect of Terms Per Query

Figures 15–20 present and contrast average transaction time, connection server utilization, and Inquery server utilization for short queries (Figures 15–17) and long queries (Figures 18–20). We also briefly discuss the effect of using medium queries. For the scaled workload, we see that query size has an even more dramatic impact on system performance when we compare the results to the single text collection architecture. Two different results are evident in these graphs. In Figures 15–17, degradations occur when the connection server becomes highly utilized. In contrast, Figures 18–20 illustrate that when clients issues long queries, the effectiveness of the architecture is poor due to bottlenecks in the Inquery servers.

Short Queries (TPQ=2) Figure 15 illustrates that until we reach 32 Inquery servers, the average transaction time improves as the number of Inquery servers increases. This result is interesting because the amount of text to search increases as the number of Inquery servers increases. For example, when the number of Inquery servers doubles, a client potentially searches twice as much information. Again, our more detailed measurements reveal that the performance improvement is due to increased parallelism during the summary commands (see Short Queries in Section 4.1.1).

In Figure 16, we see a large increase in connection server utilization as the size of the distributed system grows. At the same time, Figure 17 shows the Inquery server utilization decreases as we add Inquery servers. It is apparent that as the system size increases the connection server becomes a bottleneck causing performance to degrade. We confirmed this result by measuring the size of the message queue for the connection server. The message queue holds messages from the clients and the Inquery servers. The maximum number of outstanding messages is $|clients| + |Inquery\ servers|$ which occurs if each client and each Inquery servers sends a message to the connection server at the same time. Figure 21 shows the queue is empty for 1, 4, and 8 Inquery servers. We see that the queue length becomes extremely long and approaches 90 entries when the system contains 256 clients and 128 Inquery servers. Figure 22 shows the relationship between

Multiple Text Collections

Short Queries

TPQ	QTF	AR	TTS	TTD	DR	SO
2	Obs.	1000	30	60	1–5	1–5

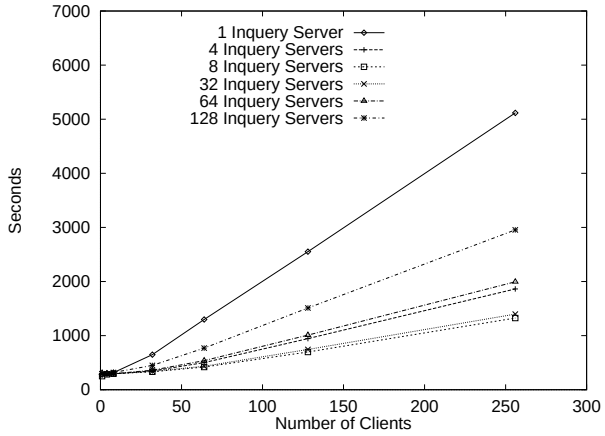

Figure 15: Average Transaction Sequence Time

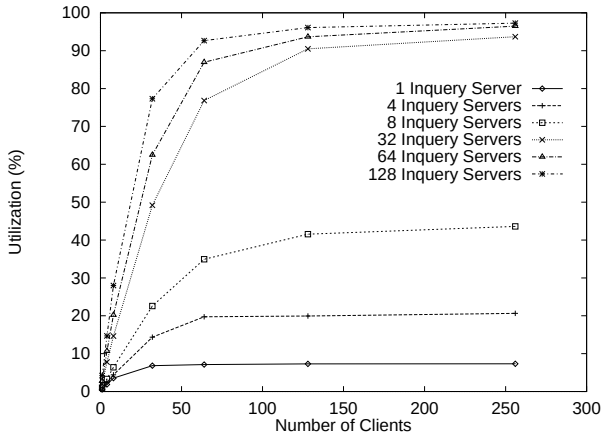

Figure 16: Connection Server Utilization

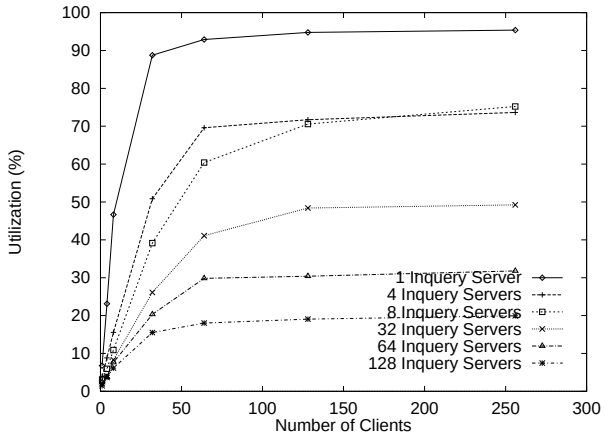

Figure 17: Average Inquery Server Utilization

Long Queries

TPQ	QTF	AR	TTS	TTD	DR	SO
27	Obs.	1000	30	60	1–5	1–5

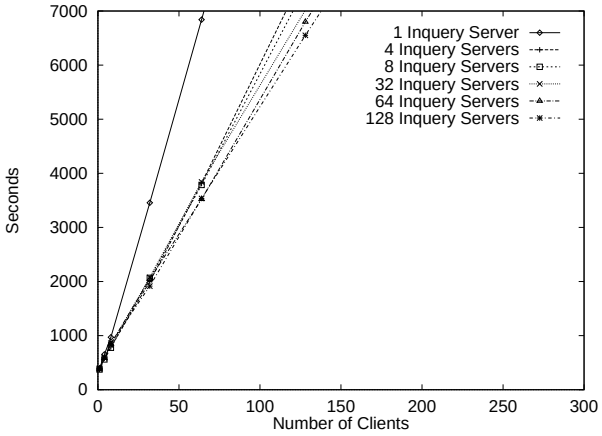

Figure 18: Average Transaction Sequence Time

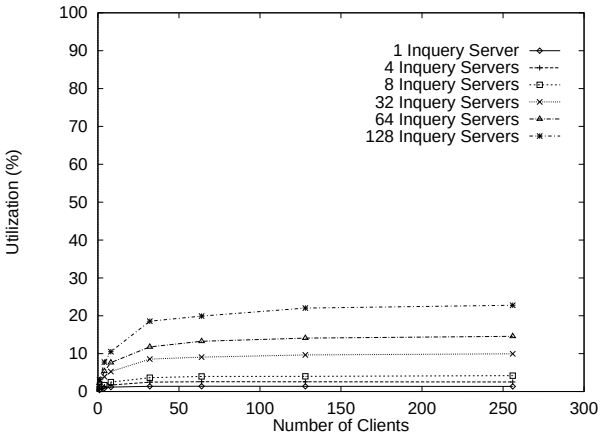

Figure 19: Connection Server Utilization

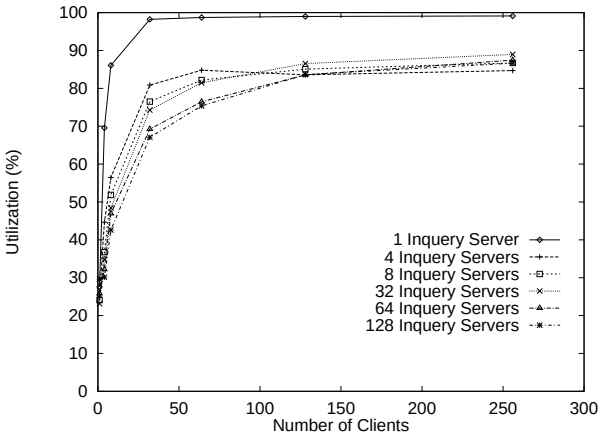

Figure 20: Average Inquery Server Utilization

TPQ	QTF	AR	TTS	TTD	DR	SO
2	Obs.	1000	30	60	1–5	1–5

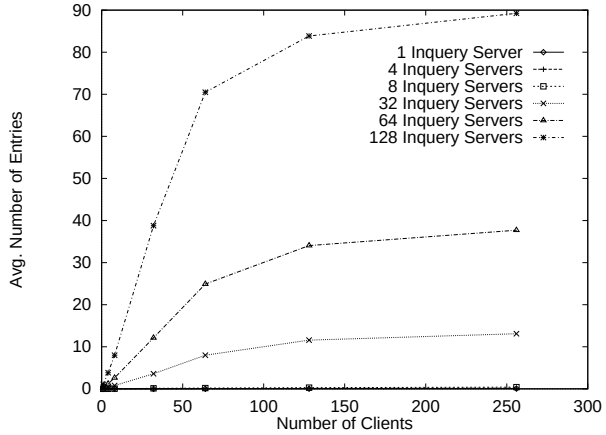

Figure 21: Avg. Length of CS Message Queue

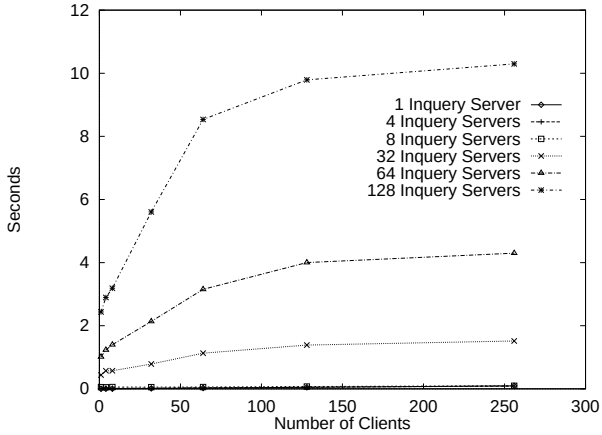

Figure 22: Avg. Time in CS Message Queue

message queue length and time. For 256 clients and 128 Inquery servers, a message spends just over 10 seconds in the message queue on average. The connection server is unable to process messages quickly enough even when the architecture contains a large number of Inquery servers but a small number of clients. For example, for a configuration with 4 clients and 128 Inquery servers, the message queue length is 3.8 entries and messages wait 2.9 seconds until the connection server is able to process them. The message delay is due to the large number of Inquery servers that send responses to the connection server.

Long Queries (TPQ=27) Figure 18 illustrates that the distributed system does not scale for long queries. In many cases, the average transaction time almost doubles as the number of number of clients doubles. The reason for the poor performance is the bottleneck in the Inquery servers (Figure 20). The Inquery server processing time and the time waiting for an available Inquery server accounts for the majority of the transaction time. Note that these values represent the average utilization over all Inquery servers. Since each client connects to a subset of the available Inquery servers, it is difficult to reach 100% average utilization. In Figure 19, we see that connection server utilization is very low. Since query evaluation dominates processing time, the connection server remains idle most of the time.

Medium Queries (TPQ=12) Our experiments indicate the Inquery servers are also the bottleneck when users issue medium length queries. Inquery server utilization using medium length queries resembles the results from Figure 20. The difference is that Inquery server utilization is slightly lower for a small number of clients, but as the number of clients increases the utilization values approach those in Figure 20. Although Inquery server utilization is similar, the effectiveness of the system when clients issue medium length queries

Multiple Text Collections

Low Query Term Frequency

TPQ	**QTF**	AR	TTS	TTD	DR	SO
27	**Low**	1000	30	60	1–5	1–5

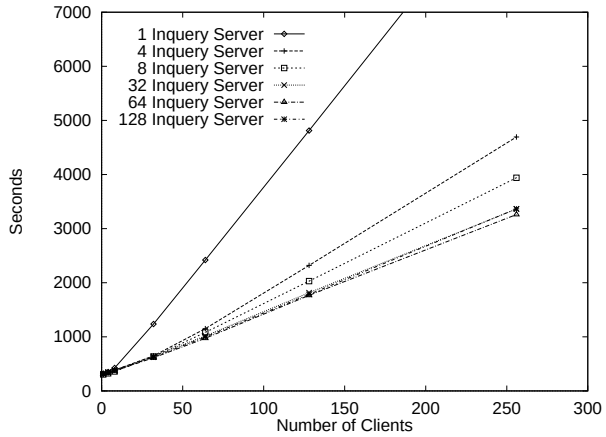

Figure 23: Average Transaction Sequence Time

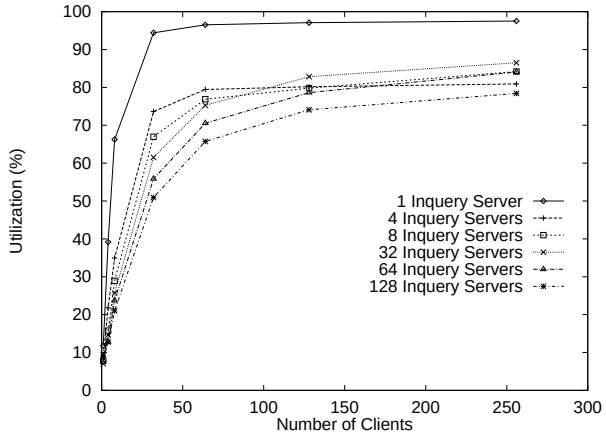

Figure 24: Average Inquery Server Utilization

High Query Term Frequency

TPQ	**QTF**	AR	TTS	TTD	DR	SO
27	**High**	1000	30	60	1–5	1–5

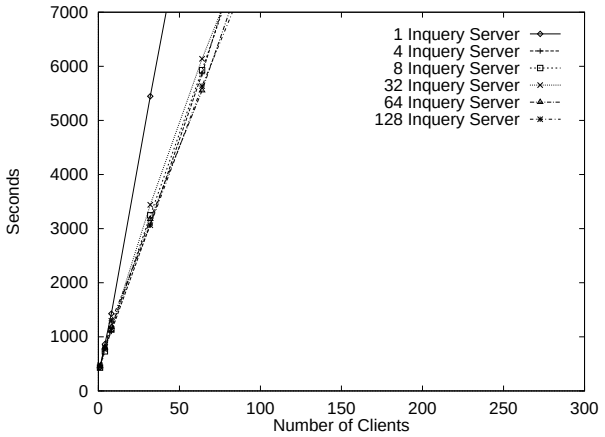

Figure 25: Average Transaction Sequence Time

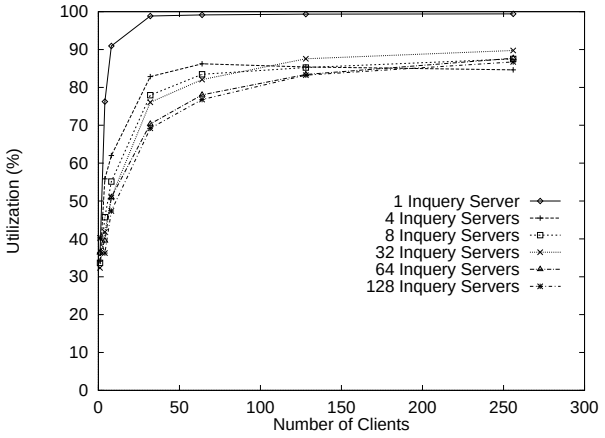

Figure 26: Average Inquery Server Utilization

is much better. Our results show that for 32, 64, 128, and 256 clients, using medium length queries improves average transaction sequence time by 54% over the transaction times when clients issue long queries. For 1, 4, and 8 clients the improvement is 13%, 40%, and 49%, respectively. We also see better connection server utilization for medium queries. The highest utilization, for 256 clients and 128 Inquery servers, approaches 62% as the number of clients increases.

4.2.2 Effect of Query Term Frequency

In Figures 23–26, we present and contrast the average transaction sequence time and Inquery server utilization for low and high skewed query term frequencies. The results are from our experiments when

clients issue long queries. In Figures 18 and 20, we show the corresponding results for the observed query term frequency. For short queries, the effect of query term frequency is not as significant since the Inquery servers are not a bottleneck.

In our retrieval engine, queries that contain terms which occur less frequently in the text collection evaluate more quickly. The opposite is true of queries containing terms which occur more frequently. Our results reflect this difference in evaluation time especially when clients issue long queries. As we expect, Figures 23 and 25 show that we obtain better performance for low skewed queries and worse performance for high skewed queries. Unfortunately, the architecture still does not scale well as we add more clients even for low skewed queries. As a comparison, the performance of the architecture with long, low skewed queries is slightly better than the performance with medium length, observed skewed queries.

Our results show the query term frequency has a large effect on transaction sequence time for long queries. Unfortunately, as Figure 23 shows, the performance is still poor even when clients issues queries with low term frequencies since Inquery server utilization is still very high. Figures 24 and 26 show that Inquery server utilization is very similar for low, observed, and high query term frequencies. However, the small differences in utilization correlate to large differences in average transaction sequence time.

4.2.3 Effect of Answers Returned

Our results indicate that returning 100, 1000, or 2000 documents has no significant effect on performance. For short queries, returning 100 documents is slightly faster than returning 1000 documents by 1.5% on average. For long queries, returning 100 documents is slightly faster by 0.16%.

The number of answers each client requests affects network utilization and the time it takes for the connection server to merge results. In our experiments, the number of answers returned is the only parameter which significantly changes network utilization. Figure 27 illustrates our architecture's typical network utilization. Utilization on a 10Mbps network is very low in our experiments. For example, the highest utilization, in Figure 27, is 3.4% using a configuration with relatively high network utilization. We do notice slightly higher network utilization for our experiments in Section 4.1 where we distribute a single text collection.

Figure 28 shows the relative network utilization when clients request 100, 1000, and 2000 answers.[3] Note in Figure 28, the x-axis is the number of Inquery servers. The relative values are the averages of all the clients. The error bars show the lowest and highest values. Figure 28 shows that as the number of clients increases, the number of answers returned has a relatively large impact on network utilization, but

[3] Figure 27 shows the actual utilization values for the 1000 answers.

TPQ	QTF	AR	TTS	TTD	DR	SO
2	Obs.	1000	30	60	1–5	1–5

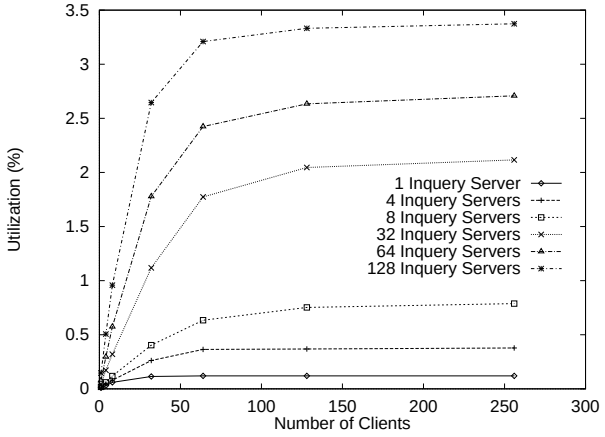

Figure 27: Network Utilization

TPQ	QTF	TTS	TTD	DR	SO
2	Obs.	30	60	1–5	1–5

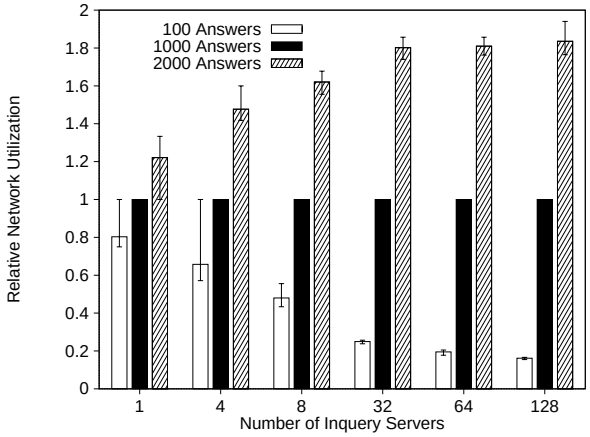

Figure 28: Effect of Answers Returned on Network Utilization

total utilization still remains low. We believe we would see a noticeable performance effect if we use a slow network or if the Inquery servers return an extremely large number of answers.

4.2.4 Effect of Think Time

Figures 29 and 30 illustrate the effect of different think times on the average transaction sequence time for short and long queries. We run our experiments using the four think time distributions from Table 1. The values in these figures are comparable to the times in Figures 15 and 18, respectively. For each client, we show the average of the relative performance values as we vary the number of Inquery servers from 1 to 128. Although the average transaction sequence time significantly varies as the number of Inquery servers change, the relative performance value does not. The error bars show the lowest and highest performance values. [4]

Figures 29 and 30 show that increasing think times results in worse system performance, especially for a small number of clients, but there is little effect as the number of clients increases. The average transaction sequence time for 128 and 256 clients is not noticeably different regardless of think time. For large system configurations, varying think time has little effect due to bottlenecks which cause the average transaction sequence to increase considerably. Time spent thinking is less of a factor when the average transaction sequence time is very large. This explains why differences in relative performance is much larger when clients issue short queries since the system achieves good performance even with a large number of clients.

[4] We do have low and high values for the solid bar (TTS=30, TTD=60) because all times are relative to those values.

IS	TPQ	QTF	AR	DR	SO
Avg.	2	Obs.	1000	1–5	1–5

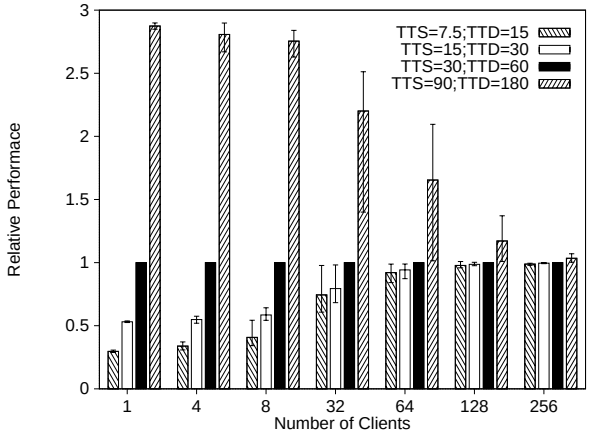

Figure 29: Effect of Think Time - Short Queries

IS	TPQ	QTF	AR	DR	SO
Avg.	27	Obs.	1000	1–5	1–5

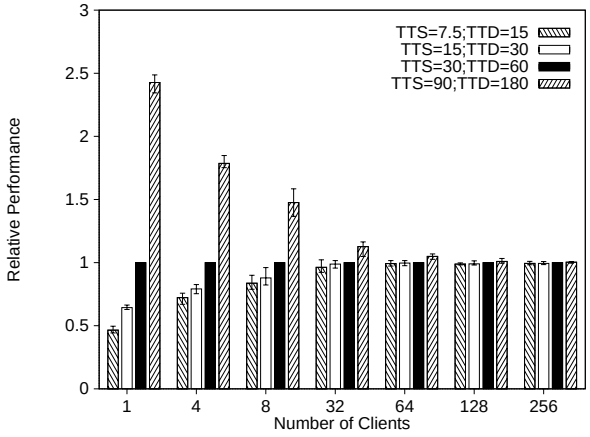

Figure 30: Effect of Think Time - Long Queries

4.2.5 Effect of Summary/Document Operations

Increasing the number of summary and document operations significantly increases the average transaction sequence time. This effect is not surprising considering each client performs more work per transaction as we increase the number of summary and document operations. Instead of comparing the average transaction sequence time, we compare response time for query, summary, and document retrieval operations occurring during a transaction sequence.

Figures 31 and 32 illustrate the effect of increasing the number of summary and document operations on response time for an architecture with 4 Inquery servers. For architectures with more than 4 Inquery servers, the effect of increasing the number of operations is even more pronounced. Figure 31 displays the results for short queries and Figure 32 displays the results for long queries. The numbers above each bar indicate the number of summary and document operations in each transaction sequence.

For a large number of clients, Figures 31 and 32 show an improvement in the response times as we increase the number of summary and document operations. For example, in both figures, the response times are very similar for 8 clients. However, for 256 clients the response times decrease as the number of summary and document retrieval operations increase. Our results also show the differences in response times is larger for long queries than for short queries.

The response time improvement occurs because the IR commands do not wait as long at the connection server. The number of outstanding requests waiting at the connection server decreases as we increase the number of summary and document operations. The decrease in the message queue length is because the average response time of the Inquery servers improves as the number of summary and document operations

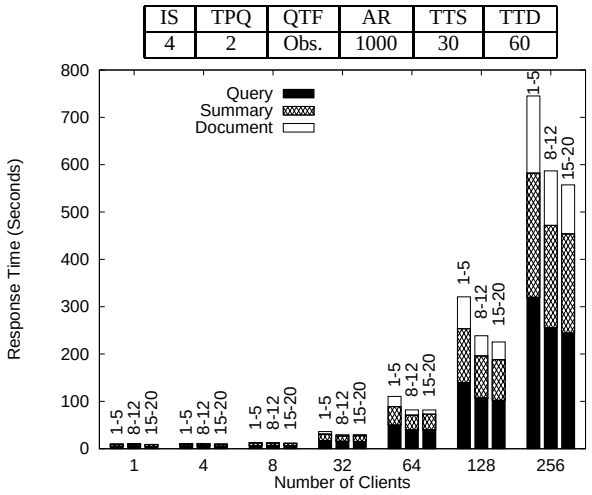

Figure 31: Effect of Summary/Document Ops. Short Queries - 4 Inquery Servers

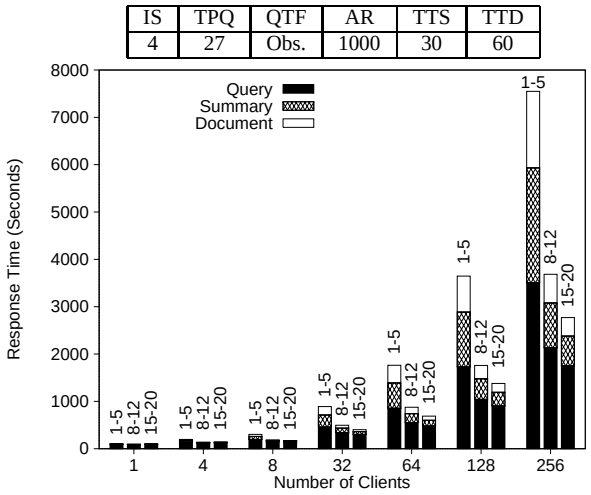

Figure 32: Effect of Summary/Document Ops. Long Queries - 4 Inquery Servers

increase. The Inquery servers are able to process summary information and document retrieval commands more quickly than query commands, especially for long queries. For example, in a configuration with 256 clients and 4 Inquery servers, an Inquery server sends an answer back to the connection server every 7.2 seconds on average when clients issue long queries and perform 1-5 summary/document operations. For 15-20 summary/documents operations, each Inquery server sends an answer back every 2.1 seconds on average. A similar effect occurs for short queries although the difference is not as large; 0.88 seconds for 1-5 summary/document operations and 0.74 seconds for 15-20.

Another factor affecting the message queue length is the rate in which clients send commands to the Inquery servers. Increasing the number of summary information and document retrieval commands per transaction sequence increases the amount of time a client spends thinking. When a client spends time thinking, the other clients are able to process requests. Also, increasing the amount of think time decreases the rate in which clients issue commands. As the clients issue commands more slowly, the number of outstanding requests for each Inquery server decreases. It is difficult for the Inquery servers to keep up with requests when clients issue commands at a fast rate, especially for query commands.

4.3 Multiple Connection Servers

Our results show that the system scales for short queries up to a certain point; if we add too many Inquery servers, then performance degrades since the connection server becomes a bottleneck. To relieve this bottleneck, we analyze the performance of a system with 2 and 4 connection servers. Including additional connection servers reduces the average utilization of each connection server, and improves performance for

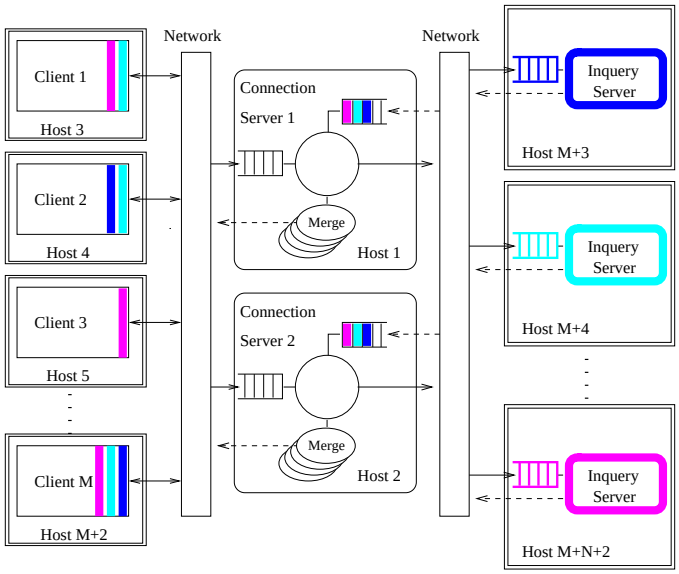

Figure 33: Multiple Connection Servers

short queries.

In this system, the clients evenly divide among the connection servers and each connection server maintains a link to all the Inquery servers. In the basic architecture, the connection server maintains a queue of outstanding requests for each of the Inquery servers. If an Inquery server is busy, the connection server adds the request to the queue. In the multiple connection server system, the connection server immediately forwards requests to the Inquery servers. Each of the Inquery servers maintains its own queue of outstanding requests. Figure 33 illustrates the new architecture.

Our experiments indicate that just moving the queues to the Inquery servers does not significantly change performance. Using short queries on the multiple text collection architecture (Section 4.2), the average performance improvement is 1.80% and the differences range from 3.7% slower to 7.7% faster. Using long queries, the performance effect ranges from 4.76% slower to 4% faster and improves by an average of 0.17%. We obtain similar results using architectures that distribute a single text collection (Section 4.1).

Our results show that including additional connection servers improves performance in large systems when users evaluate short queries. In the single connection server architecture, the connection server quickly becomes saturated with requests which limit performance. Including more connection servers eliminates the bottleneck by distributing this work. However, when the Inquery servers are the bottleneck in the system, as in Figure 18 with long queries, additional connection servers do not effect performance.

TPQ	QTF	AR	TTS	TTD	DR	SO
2	Obs.	1000	30	60	1–5	1–5

2 Connection Servers

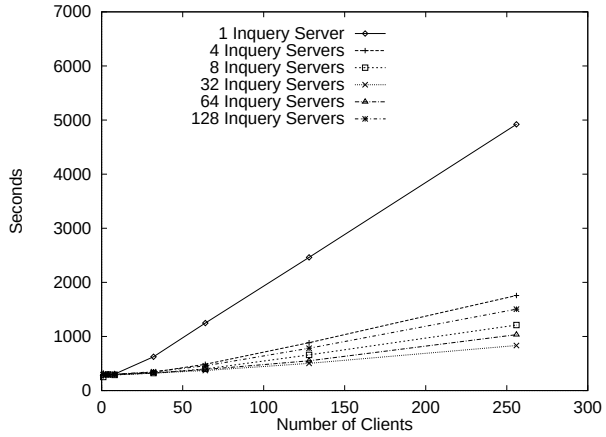

Figure 34: Average Transaction Sequence Time

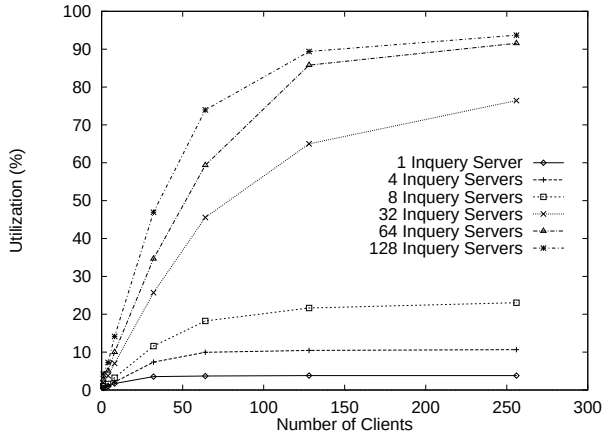

Figure 35: Avg. Connection Server Utilization

4 Connection Servers

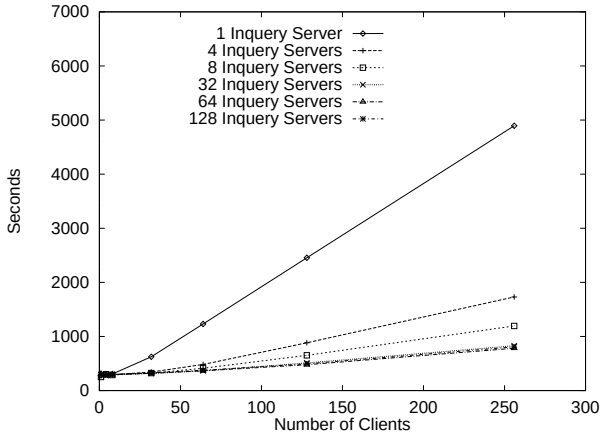

Figure 36: Average Transaction Sequence Time

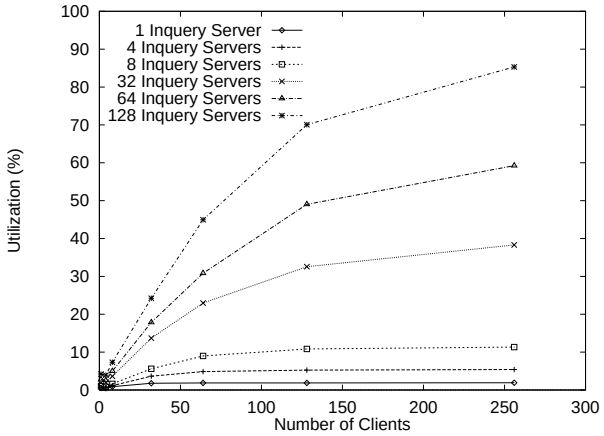

Figure 37: Avg. Connection Server Utilization

4.3.1 Two Connection Servers

Figures 34 and 35 show the average transaction sequence time and average connection server utilization for a system with two connection servers. We use short queries on an architecture with multiple text collections. Compare these results to those in Figures 15 and 16 (all of the y-axes for average transaction times are on the same scale). The best performance occurs with 32 Inquery servers.

The two connection server architecture performs better than the one server architecture as the number of clients and Inquery servers increases. For combinations of 1 to 8 clients and Inquery servers, there is not a significant difference in performance. However, for all other combinations, there is an improvement in performance. We get a speedup of 1.94 over the single connection server model for the configuration using

256 clients and 128 Inquery servers.

Figures 35 shows the reason for the speedup is due to an improvement in connection server utilization for 32, 64, and 128 Inquery servers. These are the cases where performance degrades in the 1 connection server architecture. The average decrease in utilization is 18%, 14%, and 13% for 32, 64, and 128 Inquery servers, respectively.[5] The performance of the 2 connection server architecture also degrades for large numbers of Inquery servers, but the rate is much less than the single server case. Although the connection server utilization is lower than in the 1 connection server architecture, it still gets quite high for large configurations.

4.3.2 Four Connection Servers

Figure 36 and 37 show the average transaction sequence time and average connection server utilization for a system with four connection servers. Again, for this test, we use short queries on an architecture with multiple text collections. The additional connection servers provide even greater improvements in performance for the larger configurations. We see that the best performance occurs for 32, 64, and 128 Inquery servers. This result is quite different from Figure 15 in which the performance begins to degrade after 32 Inquery servers. The most interesting effect of adding four connection servers is that the system scales very well for large configurations. We see this effect in Figure 36 where the average transaction response time of 32 to 128 Inquery servers remains nearly the same for all client configurations.

Figure 37 shows the connection server utilization is less than in the 2 connection server architecture. The maximum utilization is 87% for a configuration of 256 clients and 128 Inquery servers. The architecture is able to achieve scalable performance by managing the connection server utilization. Our results show the architecture is able to eliminate the bottleneck in the connection server by using a small number of connection servers. For short queries, 1 connection server per 32 Inquery servers enables the architecture to achieve scalable performance.

4.4 Moving Functionality

In this section, we experiment with removing the merging functionality from the connection server to reduce the amount of processing that occurs in the connection server. Results from Section 4.1 and 4.2 motivate this change since they indicate that the connection server becomes a performance bottleneck when clients issue short queries (Figures 7 – 9 and Figures 15 – 17). Our results show that moving the merging functionality *does not* significantly improve the average transaction sequence time.

[5] We exclude the case with 1 client since the results are exactly the same using 1 and 2 connection servers.

Recall that in the original prototype, the connection server is responsible for collecting and merging intermediate results before sending the final answer to the client. We change the prototype by moving the merging functionality from the connection server to the clients. Instead of merging results, the connection server immediately forwards the intermediate results to the appropriate client and each client is responsible for merging the intermediate results. The connection server now sends *N* answers to each client instead of one final answer.

A tradeoff to consider for this architecture change is the CPU time to merge results versus the CPU time to send more messages. The connection server sends more messages when the clients merge results. The following table lists the message send time versus the merge time for the cases when the clients request 100, 1000, and 2000 results. The cost for merging results or sending a message is very small, but the cost of

Operation	Answers Returned		
	100	1000	2000
Message Send	0.5 ms	2.3 ms	4.7 ms
Merge	1.9 ms	17.9 ms	35.7 ms

merging is higher than the cost of sending a message.

Table 3 shows the performance results for several configurations when we move the merging functionality. The first four rows compare results against the multiple text collection architectures (see Section 4.2). The last four rows compare results against the single text collection architecture (see Section 4.1). We focus our results on the configurations that exhibit high connection server utilization since moving the merging functionality should help these cases the most. For comparison, we show results for long queries when the connection server is not a bottleneck in rows 4 and 8.

The positive values in Table 3 indicate performance improvements due to moving the merging functionality and negative values indicate performance degradation. We show the average, minimum, and maximum values for the average transaction sequence time, the time spent in the connection server, and the connection server utilization. For example, the first row of the table shows that moving the merge functionality improves the average transaction sequence time by 0.2%. However, a negative minimum value shows that in some cases performance is worse.

The performance improvements range from 0.2% to 2.8%. We do see larger performance improvements as the clients request an increasing number of answers. The results also show larger improvements in the connection server time and utilization than in the average transaction sequence time. We do not see large improvements in performance since the amount of time it takes to perform merging is very small.

Architecture Configuration		Avg. Transaction Sequence Time			Connection Server Time			Connection Server Utilization		
		Avg.	Min	Max	Avg.	Min	Max	Avg.	Min	Max
Multiple Collections	Short Queries 100 Answers	0.2%	-3.5%	5.8%	3.4%	0.4%	7.3%	3.2%	-4.4%	9.8%
	Short Queries 1000 Answers	1.0%	-8.0%	6.9%	5.3%	2.2%	7.8%	4.3%	-0.3%	12.3%
	Short Queries 2000 Answers	1.3%	-5.5%	8.2%	6.9%	4.6%	9.2%	5.6%	-0.3%	11.7%
	Long Queries 1000 Answers	-0.5%	-4.6%	3.3%	5.2%	1.7%	8.4%	5.6%	1.3%	9.8%
Single Collection	Short Queries 100 Answers	0.2%	-11.2%	5.8%	2.5%	-1.8%	6.5%	2.2%	-6.6%	9.8%
	Short Queries 1000 Answers	1.6%	-5.1%	6.2%	5.0%	1.5%	7.4%	3.4%	-0.4%	9.6%
	Short Queries 2000 Answers	2.8%	-4.5%	9.9%	7.3%	2.8%	11.0%	4.5%	0.0%	9.9%
	Long Queries 1000 Answers	0.5%	-3.9%	5.6%	5.2%	-3.9%	7.8%	4.7%	0.4%	9.4%

Table 3: Performance Effects from Moving Merging

Furthermore, the time it takes for the connection server to send more messages limits the time savings from removing the merging functionality in the connection server. One possible solution is to have the Inquery servers return answers directly to the clients rather than having to go through the connection server.

5 Related Work

Related work on architectures for distributed IR systems include designing and evaluating architecture performance, data partitioning, caching, and multiprocessor systems. The work on architecture performance most closely relates to this work. Our research combines and extends previous work in distributed IR since we model and analyze a complete system architecture. Although others have examined some of the issues, no one has considered the entire system under a variety of workloads and conditions. We also experiment with very large text collections; up to 128 GB of data and 256 users. Prior work has not examined such large systems. Furthermore, we base our distributed system on a proven, effective retrieval engine.

Burkowski reports on a simulation study which measures the retrieval performance of a distributed IR system [Burkowski, 1990]. The experiments explore two strategies for distributing a fixed workload across a small number of servers. The first equally distributes the text collection among all the servers. The second splits servers into two groups, one group for query evaluation and one group for document retrieval. This work is the most closely related to our work, but differs in several ways. He assumes a worst case workload where each user broadcasts queries to all servers without any think time. We experiment with

larger distributed configurations, we vary the number of clients, and use a more realistic user workload which uses both query and document retrieval operations.

Lin and Zhou implement a distributed IR system on a network of DEC5000 workstations using PVM (Parallel Virtual Machine) to coordinate work and network communication [Lin and Zhou, 1993]. They develop a new retrieval model that uses a variation of the signature file encoding scheme to map document collections over the network. Their results show large speedup improvements due to parallelization. Our research extends this work by analyzing a distributed system to increase efficiency. Since we use a simulator, we are also able to run more experiments under diverse conditions. Another advantage of our work is that we base our distributed system upon a proven and effective retrieval model rather than using a new model developed to expose parallelism.

Couvreur *et al.* analyze the performance and cost factors of searching large text collections on parallel systems [Couvreur et al., 1994]. They use simulation models to investigate three different hardware architectures and search algorithms including a mainframe system using an inverted list IR system, a collection of RISC processors using a superimposed IR system, and a special purpose machine architecture that uses a direct search. The focus of the work is on analyzing the tradeoff between performance and cost. Their results show that mainframe configuration is the most cost effective. They also suggest that using an inverted list algorithm on a network of workstations would be beneficial but they are concerned about the complexity. Our research shows the effectiveness of using a network of workstations. We evaluate the performance of different configurations and explore improvements. Under realistic workloads, we present a scalable architecture for distributed information retrieval.

Other researchers have also designed distributed IR architectures [Harman et al., 1991, Danzig et al., 1991, Sheldon et al., 1994, Crowder and Nicholas, 1995]. Our contribution is that we show how different system parameters affect performance and scalability of a distributed IR system. Also, we show that a simple distributed architecture performs well under large, realistic configurations.

Several studies have examined the use of caching in distributed IR systems [Martin et al., 1990, Martin and Russell, 1991, Simpson and Alonso, 1987, Tomasic and Garcia-Molina, 1992, Schatz, 1990]. The client caches data so that operations are not repeatedly sent to the remote server. Instead, the client locally performs frequent operations. The use of caching is most beneficial for systems that are distributed over slow networks or that evaluate queries slowly. We do not study caching effects since we implement our system on a fast local network. Furthermore, implementing a system that performs caching at the client requires a different retrieval engine.

Other researchers have investigated various data partitioning schemes for distributed IR systems [Tomasic, 1994,

Tomasic and Garcia-Molina, 1993, Macleod et al., 1987, Jeong and Omiecinski, 1995]. We briefly address this issue in the experiments in Section 4.1. Although we only consider one partitioning scheme, we extend previous results in several ways. Our experiments include results for both small and large configurations. Previous research has investigated only small configurations. We investigate changes to the architecture that do not involve changes to the underlying retrieval model. Several of the partitioning schemes mentioned in the previous work require changes to the retrieval model which possibly alters retrieval effectiveness.

The performance of our distributed system relies upon the amount of parallelism we are able to exploit. Beginning in the late 1980's several researchers implemented IR systems on multiprocessor machines [Pogue and Willett, 1987, Stanfill and Kahle, 1986, Stanfill et al., 1989, Cringean et al., 1990, Frieder and Siegelmann, 1991 Bailey and Hawking, 1996]. We do not investigate multiprocessor systems in our experiments. Instead, we concentrate on a system in which each component of the architecture is independent. Our architecture is able to use a network of workstations instead of relying on special hardware. As future work, we will investigate issues such as the performance of our system on shared memory multiprocessors and multiprocessing.

6 Summary

To keep pace with the increasing amounts of online information, the performance of information retrieval systems must improve. In this paper, we present an implementation of a distributed IR system to achieve coordinated, concurrent, and scalable access. We develop a flexible simulation model to examine the performance of the prototype using a wide variety of parameters, workloads, and configurations. We present results that measure system response time, utilization, and identify bottlenecks.

We present results for a system that distributes a single text collection across a number of servers. We vary the number of users and text collections, and show that the number of terms per query has a significant effect on performance and utilization. We also present more detailed results of the performance of a system that distributes multiple text collections. These results show the effect of varying the query term frequencies, answers returned, think time, and the number of summary and document operations as well as the number of users and text collections and the number of terms per query.

The best performance occurs for architectures with either 8 or 32 Inquery servers when the system is able to exploit parallelism in the summary information commands. Performance degrades for more than 32 Inquery servers when the connection server becomes a bottleneck, especially for short queries. Based upon these results we change the architecture to improve performance and scalability. One change involves moving the merging functionality from the connection server to the clients. Our results show that this change

has very little impact on performance. Our second change involves adding a small number of connection servers. Our results show that an architecture with only 1 connection server per 64 clients and 32 Inquery servers achieves scalable performance when clients send short queries. Short queries are a realistic workload, since several studies of existing IR systems demonstrate that users tend to use short queries. By adding a small number of connection servers to coordinate a large number of clients and Inquery servers, the system maintains scalable performance at higher workloads.

When the system bottleneck is the Inquery server, as for long queries, it is more difficult to achieve reasonable performance. As future research, we will investigate replicating the collections, shared-memory multiprocessing, and multithreading. Replication requires additional functionality in the connection server for coordinating access and load balancing. Using a multiprocessor should provide parallel access without paying the resource costs of replication. However, the high I/O demands of information retrieval may overwhelm a shared-memory multiprocessor. We are also investigating multithreading for the connection servers and Inquery servers. A multithreaded Inquery server will improve response times and resource utilization.

Acknowledgments

We would like to thank Bob Cook and Kathleen Dibella for help with the development of the prototype system. We also thank Jamie Callan, Bruce Croft, and Zhihong Lu for their contributions to this work.

References

[Bailey and Hawking, 1996] Bailey, P. and Hawking, D. (1996). A parallel architecture for query processing over a terabyte of text. Technical Report TR-CS-96-04, The Australian National University.

[Brumfield et al., 1988] Brumfield, J. A., Miller, J. L., and Chou, H.-T. (1988). Performance modeling of distributed object-oriented database systems. In *1988 International Symposium On Databases in Parallel and Distributed Systems*, pages 22–32, Austin, TX.

[Burkowski, 1990] Burkowski, F. J. (1990). Retrieval performance of a distributed text database utilizing a parallel process document server. In *1990 International Symposium On Databases in Parallel and Distributed Systems*, pages 71–79, Trinity College, Dublin, Ireland.

[Cahoon and McKinley, 1996] Cahoon, B. and McKinley, K. S. (1996). Performance evaluation of a distributed architecture for information retrieval. In *Proceedings of the Nineteenth Annual International ACM*

SIGIR Conference on Research and Development in Information Retrieval, pages 110–118, Zurich, Switzerland.

[Callan et al., 1995a] Callan, J. P., Croft, W. B., and Broglio, J. (1995a). TREC and TIPSTER experiments with INQUERY. *Information Processing & Management*, 31(3):327–343.

[Callan et al., 1992] Callan, J. P., Croft, W. B., and Harding, S. M. (1992). The INQUERY retrieval system. In *Proceedings of the 3rd International Conference on Database and Expert System Applications*, Valencia, Spain.

[Callan et al., 1995b] Callan, J. P., Lu, Z., and Croft, W. B. (1995b). Searching distributed collections with inference networks. In *Proceedings of the Eighteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, Seattle, WA.

[Couvreur et al., 1994] Couvreur, T. R., Benzel, R. N., Miller, S. F., Zeitler, D. N., Lee, D. L., Singhai, M., Shivaratri, N., and Wong, W. Y. P. (1994). An analysis of performance and cost factors in searching large text databases using parallel search systems. *Journal of the American Society for Information Science*, 7(45):443–464.

[Cringean et al., 1990] Cringean, J. K., England, R., Mason, G. A., and Willett, P. (1990). Parallel text searching in serial files using a processor farm. In *Proceedings of the Thirteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, Brussels, Belgium.

[Croft et al., 1995] Croft, W. B., Cook, R., and Wilder, D. (1995). Providing government information on the internet: Experiences with THOMAS. In *The Second International Conference on the Theory and Practice of Digital Libraries*, Austin, TX.

[Crowder and Nicholas, 1995] Crowder, G. and Nicholas, C. (1995). An approach to large scale distributed information systems using statistical properties of text to guide agent search. In *CIKM Workshop on Intelligent Information Agents*, Baltimore, MD.

[Danzig et al., 1991] Danzig, P. B., Ahn, J., Noll, J., and Obraczka, K. (1991). Distributed indexing: A scalable mechanism for distributed information retrieval. In *Proceedings of the Fourteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 221–229, Chicago, IL.

[Fox, 1983] Fox, E. A. (1983). Characterization of two new experimental collections in computer and information science containing textual and bibliographic concepts. Technical Report 83-561, Cornell University, Ithaca, NY.

[Frieder and Siegelmann, 1991] Frieder, O. and Siegelmann, H. T. (1991). On the allocation of documents in multiprocessor information retrieval systems. In *Proceedings of the Fourteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 230–239, Chicago, IL.

[Harman, 1992] Harman, D., editor (1992). *The First Text REtrieval Conference (TREC-1)*. National Institute of Standards and Technology Special Publication 200-217, Gaithersburg, MD.

[Harman et al., 1991] Harman, D., McCoy, W., Toense, R., and Candela, G. (1991). Prototyping a distributed information retrieval system that uses statistical ranking. *Information Processing & Management*, 27(5):449–460.

[Jeong and Omiecinski, 1995] Jeong, B.-S. and Omiecinski, E. (1995). Inverted file partitioning schemes in multiple disk systems. *IEEE Transactions on Parallel and Distributed Systems*, 6(2):142–153.

[Jump, 1993] Jump, J. R. (1993). *YACSIM Reference Manual*. Rice University, version 2.1.1 edition.

[Lin and Zhou, 1993] Lin, Z. and Zhou, S. (1993). Parallelizing I/O intensive applications for a workstation cluster: a case study. *Computer Architecture News*, 21(5):15–22.

[Macleod et al., 1987] Macleod, I. A., Martin, T. P., Nordin, B., and Phillips, J. R. (1987). Strategies for building distributed information retrieval systems. *Information Processing & Management*, 23(6):511–528.

[Martin et al., 1990] Martin, T. P., Macleod, I. A., Russell, J. I., Lesse, K., and Foster, B. (1990). A case study of caching strategies for a distributed full text retrieval system. *Information Processing & Management*, 26(2):227–247.

[Martin and Russell, 1991] Martin, T. P. and Russell, J. I. (1991). Data caching strategies for distributed full text retrieval systems. *Information Systems*, 16(1):1–11.

[Moffat and Zobel, 1995] Moffat, A. and Zobel, J. (1995). Information retrieval systems for large document collections. In Harman, D. K., editor, *The Third Text REtrieval Conference (TREC-3)*, pages 85–94. National Institute of Standards and Technology, Special Publication 500-225.

[Pogue and Willett, 1987] Pogue, C. A. and Willett, P. (1987). Use of text signatures for document retrieval in a highly parallel environment. *Parallel Computing*, 4:259–268.

[Schatz, 1990] Schatz, B. R. (1990). Interactive retrieval in inforamtion spaces distributed across a wide-area network. Technical Report TR 90-35, University of Arizona, Dept. of Computer Science.

[Sheldon et al., 1994] Sheldon, M. A., Duda, A., Weiss, R., James W. O'Toole, J., and Gifford, D. K. (1994). Content routing for distributed information servers. In *Proc. Fourth International Conference on Extending Database Technology*, Cambridge, England.

[Simpson and Alonso, 1987] Simpson, P. and Alonso, R. (1987). Data caching in information retrieval systems. In *Proceedings of the Tenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 296–305, New Orleans, LA.

[Stanfill and Kahle, 1986] Stanfill, C. and Kahle, B. (1986). Parallel free-text search on the connection machine system. *Communications of the ACM*, 29(12):1229–1239.

[Stanfill et al., 1989] Stanfill, C., Thau, R., and Waltz, D. (1989). A parallel indexed algorithm for information retrieval. In *Proceedings of the Twelfth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 88–97, Cambridge, MA.

[Tomasic, 1994] Tomasic, A. (1994). *Distributed Queries and Incremental Updates In Information Retrieval Systems*. PhD thesis, Princeton University.

[Tomasic and Garcia-Molina, 1992] Tomasic, A. and Garcia-Molina, H. (1992). Caching and database scaling in distributed shared-nothing information retrieval systems. Technical Report STAN-CS-92-1456, Stanford University.

[Tomasic and Garcia-Molina, 1993] Tomasic, A. and Garcia-Molina, H. (1993). Performance of inverted indices in shared-nothing distributed text document information retrieval systems. In *Proceedings of the Second International Conference on Parallel and Distributed Information Systems*, San Diego, CA.

[Viles and French, 1995] Viles, C. L. and French, J. C. (1995). Dissemination of collection wide information in a distributed information retrieval system. In *Proceedings of the Eighteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, Seattle, WA.

[Voorhees et al., 1995] Voorhees, E. M., Gupta, N. K., and Johnson-Laird, B. (1995). Learning collection fusion strategies. In *Proceedings of the Eighteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, Seattle, WA.

[Wolfram, 1992] Wolfram, D. (1992). Applying informetric characteristics of databases to IR system file design, part I: Informetric models. *Information Processing & Management*, 28(1):121–133.

[Zipf, 1949] Zipf, G. K. (1949). *Human Behavior and the Principle of Least Effort*. Addison-Wesley Press.